Topic 4

Feature extraction from Time Series data [TS]

4.1 Introduction

Space and time are the fundamental concepts with which we build our own reality. Hence, spatial and time series data are extensively available in every scientific and application domain. Typical examples include weather data in meteorology, stock market data in economics, energy data in industries, physiological signals such as ECG/EEG in medicine. In general, these spatio-temporal data are analyzed either to understand the underlying mechanism generating the data, to find patterns for inferring high level concepts (i.e., classification or clustering) or to predict the future behaviors of the systems based on their past data.

4.1.1 Global description of the practicum and project

This practicum will focus mainly on key techniques for analyzing time-series data such as peak detection, filters, Fourier analysis (FFT), Dynamic Time Warping (DTW) and prediction models.

4.1.2 Study material and tools

Although one can program in Python with just an editor and a Python interpreter, programming in an IDE (Integrated Development Environment) has many advantages. We recommend PyCharm.[1] Also Spyder[2] seems very nice. Also, we recommend Anaconda[3] distribution to collectively install the required Python packages such as NumPy, Pandas, matplotlib, scikit-learn. Section 4.2.8 gives a short tutorial on the necessary basic concepts in Python. Note that the tutorial is optional and can be safely skipped by those who have sufficient knowledge in Python.

The following reading materials are suggested for different topics,

1. Time domain features
   (a) `http://pandas.pydata.org/pandas-docs/stable/api.html#api-dataframe-stats`

[1] `https://www.jetbrains.com/pycharm/`
[2] `https://www.spyder-ide.org/`
[3] `https://docs.continuum.io`

   (b) https://docs.scipy.org/doc/scipy-0.7.x/reference/stats.html

2. Frequency domain features
   (a) Programatically Understanding Series
   (b) Frequency Domain and Fourier Transforms
3. Dynamic time warping
   (a) http://www.phon.ox.ac.uk/jcoleman/old_SLP/Lecture_5/DTW_explanation.html
   (b) Programatically Understanding Series (section time warping)
4. Time series prediction
   (a) http://www.ulb.ac.be/di/map/gbonte/ftp/time_ser.pdf
   (b) https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/
5. Using Python for data analysis

   There material goes beyond the scope of this topic but might be interesting for advanced readers,
   (a) https://github.com/icounter/python-data-mining-and-text-mining/blob/master/Python4DataAnalysis.pdf
   (b) https://www.youtube.com/user/sentdex/playlists?shelf_id=5&view=50&sort=dd

4.1.3 Deliverables and obligatory items

The practicum requires individual files including images, results and source code to be combined together in a single PDF and upload to the canvas. For each question, the source code and console outputs can be put together in a single text file. Also, add comments and explanations as needed to support the code and results.

4.2 Description of the practical assignments

In this practicum, we focus mainly on two topics, i.e., classifying time series data and time series prediction. We begin with simple questions to describe and better understand the dataset recommended for this topic. Then, we explore key techniques such as peak detection, filters, Fourier analysis (FFT), Dynamic Time Warping (DWT) and correlation analysis.

4.2.1 Datasets

We work with the following data sets,

1. **UCI HAR activity recognition data set**. For classifying time series data, we will work with an activity recognition data from inertial sensors (e.g., accelerometer and gyroscope). It has both raw data and common derived features. Also, a youtube video gives an overview of the dataset.
   https://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones
   The basic description about the dataset and the files are given in the README.txt
2. **Berkeley Earth Surface Temperature data set**.

   For time series prediction, we will work with data from the Berkeley Earth Surface Temperature Study.
   https://www.kaggle.com/berkeleyearth/climate-change-earth-surface-temperature-data
   This extensive data set is created by combining multiple existing archives and it allows groupings by country or city of interest.

Feature data (X)	Labels (Y)
4 (wheels), 5 (doors), 1200 (kilograms)	1 (Car)
6, 2, 6000	1
4, 3, 900	0
4, 2, 7800	1

Table 4.1: Illustration of feature and label data.

4.2.2 Understanding the dataset

30 minutes

The first dataset we will be working with is the UCI HAR Dataset. It is a dataset that contains recorded signals of human movement. More specifically, several persons have performed six activities (WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS, SITTING, STANDING, LAYING) whilst wearing a smartphone (Samsung Galaxy S II) on the waist. The embedded accelerometer and gyroscope of this smartphone were used to capture the 3-axial linear acceleration and 3-axial angular velocity. The UCI dataset includes both the raw signals obtained from these sensors, as well as a derived feature vector from these signals. This is a good point to first read the `README.txt` file from the dataset.

Features

For those unfamiliar with machine learning and feature engineering in particular; a feature is a measurable property or distinct piece of information about that what we are observing or measuring. For more intuition, let's consider a very common machine learning task; *classification*. In classification, we have a set of observations which we want to place in a distinct class. Suppose we have a collection of vehicles which contains cars and trucks and we want to place them in the right category or class, namely *car* or *truck*. For us this would be simple, since we know the differences between a car and a truck. But a computer generally does not, and we have to provide it with pieces of information about cars and trucks so that it can learn the difference between them and classify them. These pieces of information are features. For a successful classification it is important that we choose the features such that they are both informative and discriminative, the latter is especially important since we want to discriminate between two classes. A possible feature could be the number of wheels. This feature is likely not discriminative since most cars have four wheels and most trucks have four wheels as well (some may have six). Perhaps a better feature would be the weight of the vehicle. The weight of a truck is generally much higher than that of a car. Another feature could be the number of doors; a truck generally has two doors and a car at least three. You can imagine that we can come up with much more features and in general, the more informative and discriminative features we come up with (for now we ignore the fact that features may interfere with each other), the better the computer can differentiate between trucks and cars and thus the better the classification will be.

The authors of the UCI HAR dataset have derived 561 (!) features from the raw signals and later in this assignment, we are going to try to classify the signals to one of the six categories, which are the activities.

Training and Testing Labels

During the training phase, you need train data that describes the observation of our vehicle in terms of features, but you also need to tell your model whether this observation and these features belong to a truck or a car. The ground truth about the observation is called the class label. Labels are often numeric for classification problems, so a 0 denotes a car and a 1 denotes a truck. In many machine learning problems, the feature data is often denoted with an X, and the corresponding labels are often denoted with Y, following that X generally considers a variable or an input, and Y is the output or result of a function. See Table 4.1 for an example.

The UCI HAR dataset handles the same nomenclature. The file `X_train` contains the feature data and `Y_train` contains the corresponding labels, only we have 6 activities or classes instead of two, so the labels range from 1 to 6. The same logic goes for the test data.

Pre-Processing and Windowing

The creation of features from raw data requires various pre-processing techniques. An excerpt from the README of the UCI HAR dataset:

> "*The sensor signals (accelerometer and gyroscope) were recorded with a sampling rate of 50 Hz and pre-processed by applying noise filters and then sampled in fixed-width sliding windows of 2.56 sec and 50% overlap (128 readings/window).*"

We will go into more depth about filtering later in this assignment, but for now we will first focus on the windowing mentioned here. The time series data is non-stationary, which means that the statistical properties of the signal change over time. However, when you derive features you want to do so from a part of the signal that is stationary. So instead of computing features over the whole signal which is non-stationary, you take very small windows (or blocks) of a signal (in the order of tens of milliseconds) in which you assume that the signal in this window is stationary. Both the raw data (found in the folders `Inertial Signals`) and feature data of this dataset consists of these windows. The length of each window of this dataset is 2.56 seconds. The overlap is 50% (observe that for any file in folders `Inertial Signals` first *first 50%* of the values in *row 2* are the same as the *last 50% values in row 1*).

☞ **4.1** Now, it is your turn to get to know the dataset better. For these assignments you can use the dataset with the derived features, located in `train/X_train.txt` and `test/X_test.txt`.

Refer to the tutorial in [1] for some useful built-in methods from the Python library *Pandas*. The following concepts/functions are useful:

- **Reading Files:** `read_fwf`. Note that the data is separated by a space not by a comma or tab therefore you might need the optional argument `delim_whitespace=True`. Moreover, the file does not have any headers.
- **Data Frame Dimension:** `shape` function from *Pandas*.
- **Sub-setting data frame**: It can be done in various ways. Since the data is in the form of rows and columns so easiest is to use `iloc`.
- **Basic Statistics:** You can use function such as `mean, median, max, min, std`. Alternatively, you can use `describe`.

You may want to combine features names with their respective values. Note that the names of feature are stored in a separate file named `features`

(a) Describe the dataset in terms of dimensionality, how many columns and rows are there? What do the rows and columns represent?
(b) Give summary statistics for some features of your choice (at least 5). Examples of statistics are means, standard deviations and upper and lower percentiles.

□

We now consider the ground truth class labels. These are located in `train/Y_train.txt` and `test/Y_test.txt` respectively.

☞ **4.2** For creating Bar plots there are at least two ways in Pandas

- **Bar Chart:** You need to manually calculate the counts for each activity type. You can do it either by using a for loop or using a combination of `groupby` and `count`. Finally, create bar plot using `bar`
- **Histogram** `hist` to create a Histogram.

(a) Describe this file in terms of dimensionality, how many columns and rows are there? What do they represent (this is a very obvious question of course, don't over-think it)?
(b) Plot the number of datapoints for each user activity (output class) as a bar chart. Is the dataset balanced? Why is it important, for machine learning and classification purposes, that a dataset is balanced?

□

Next, we want to compare the values and shapes of the features for every class. The features may look different for different classes (a truck has 2 doors and a car 3 or more) and this corresponds to the idea that we want to differentiate and classify whether a signal from this database comes from walking or standing for example. To be able to do this, you need to couple the class labels (Y) with the feature data values (X). You have inspected the dimensions of both which provides a clue about how this should be done.

To create distribution you can use the following:

- **Features to Distribution:** The `seaborn.kdeplot` package in Python allows you to easily use *Kernel Density Estimates* (KDE) to turn the feature values into distributions.

☞ **4.3** Once you have isolated the feature values for each class choose 10 features (it does not matter which ones). For every feature, plot the distributions of every class for that feature in one figure. So, for a single feature you should have one plot which includes six distributions corresponding to the feature values belonging to those classes. Visually inspect the six different distributions in the figure for every feature and report the nature of distributions (e.g., normal distribution or not, skewness, modality, etc.). Is it possible to discriminate between activities based on those chosen features? Are there differences between the walking activities and non-walking activities such as lying down and sitting? What explains these differences? □

4.2.3 Exploring the Raw Data

30 minutes

For this part of the assignment, we are going to focus on the raw signal sensor data of the UCI HAR dataset. The raw data can be found in the folders `/Inertial Signals` in the `train` and `test` folders respectively. You will find 6 files which correspond to the axes of the accelerometer and gyroscope respectively.

Calculate the variances of the accelerations of all 3 axes with aid of the `total_acc_` files of the 3 axes and select the corresponding body acceleration file (`body_acc_`) from the axis that has the highest variance. [see also question below]

The raw data is somewhat different from the feature data. In the feature data, every feature had 1 value which was calculated from each window, thus the whole window is represented as 1 value. In the raw data, the windowing is again used but since no features have been created from this raw data, this data contains the individual datapoints from the windows. We first want to reconstruct the original signal from this windowed version.

☞ **4.4** The following functions from pandas shall be useful

- To catenate columns above each other you can use `melt`. For an example, see here also.

(a) Calculate the variances of the accelerations of all 3 axes with aid of the `total_acc_` files of the 3 axes and select the corresponding body acceleration file (`body_acc_`) from the axis that has the highest variance.

(b) Describe this axis file in terms of dimensionality, how many columns and rows are there? What do they represent in this case? **Hint:** Use the information about the sample rate and window size to determine the answers.

(c) Again, couple the class labels (Y) with the raw data points (X). You have to keep in mind that the windows have 50% overlap; what is the consequence of this overlap and how do you go about this? Once you have successfully coupled the class labels with the windows and dealt with the window overlap, you can concatenate the datapoints to reconstruct the original signal. You do not have to repeat this process for all files, you can use this piece of raw signal in assignments that require you to work with raw signal.

□

4.2.4 The Time Domain and the Frequency Domain

Features can be constructed from the signal in the time domain or from the frequency domain. First we will start to look at the signal in the frequency domain.

☞ **4.5**

(a) Use your raw signal to recreate some time domain features such as mean, standard deviation, kurtosis or features of your own choosing; some examples can be found in the article by Altin et al. [2].
(b) Report whether some of the time domain features have the potential to discriminate between multiple activities. For example, you could combine the means and standard deviations of the classes to reconstruct a normal distribution of the classes and visually inspect whether discrimination between activities is possible.

□

In some cases, looking at a signal in the time domain is not informative. Looking at your raw signal, you will see a significant amount of noise and at first glance, any pattern is difficult to discover. Another way of looking at the signal is in terms of the frequencies it consists of. Decomposing the signal into frequencies and magnitudes of those frequencies can show useful patterns (dominating frequencies) in the frequency domain. One technique to translate a signal towards the frequency domain is the *Fourier transform*. You can read the excerpt of Jurafsky and Martin (section 9.3.3) [3] for a quick refresh. You can watch the video by Wiliam Cox in reference [4] if you need more background on this subject. In addition, Kulkarni [5] provides the more theoretical background. The section in Jurafsky and Martin covers why we want to use windowing as well as the notion that we will use a special case of the Fourier transform; the Discrete Fourier Transform (DFT) for discrete sampled data.

An effective algorithm to compute the DFT is called the Fast Fourier Transform (FFT). Python's `scipy` module comes with the option to calculate the FFT of a signal with the `fft` method. This method returns the Fourier coefficients which contain a real and an imaginary part since the transfrom is based on Eulers formula. The real and imaginary part correspond to the magnitude and phase of the frequency components present in the signal. To correctly plot a spectrum, some more steps are required and these are explained in [6]. This reference also covers how the symmetry of the transform can be handled to correctly plot the resulting spectrum. Alternatively, some more information and a more practical approach to programming the FFT is given in the Python project by Punbat [7].

☞ **4.6** Read the scipy documentation to complete the following questions.

(a) Implement the Fourier transform to plot the spectra of all 6 activities. Keep in mind the length of the sample you want to transform to the frequency domain and the sample rate (referred to as sample spacing in the `scipy` documentation). It could be that a large peak is visible in the spectrum at 0Hz. A frequency of 0 Hz denotes something that is referred to as a DC-offset. You can remove a DC-offset by subtracting the mean of the data from the data as `data=data-mean(data)`. It is wise to create a test signal with known frequency components and amplitudes to test your implementation of the Fourier transform:

$$sin(2*2\pi x)+0.8sin(4*2\pi x)+0.5sin(6*2\pi x)$$

This signal has components of 2 Hz, 4Hz and 6Hz with amplitude 1.0 , 0.8 and 0.5 respectively. See Figure 4.1 for plots of the test signal and its spectrum.
(b) After plotting the spectra for all 6 activities, determine whether it is possible to discriminate between activities based on the spectra.

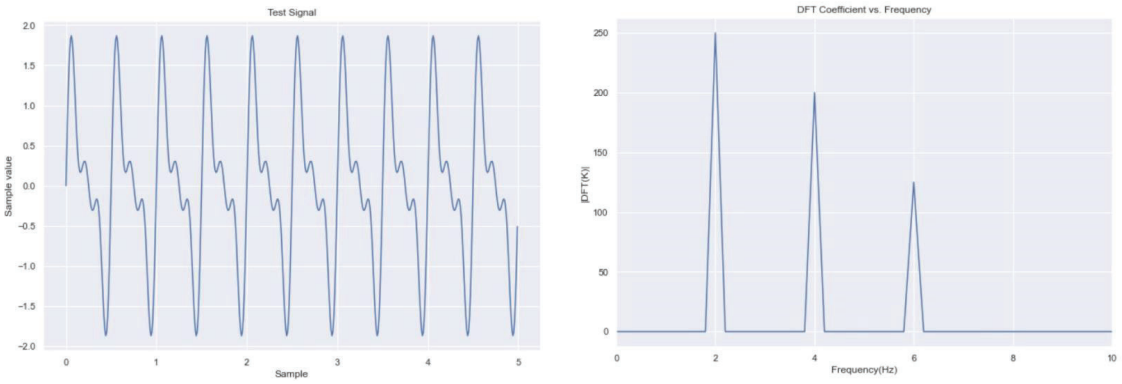
□

4.2.5 Filtering

Accelerometer data is fairly noisy. Especially the high frequency components interfere with the signals of our interest (i.e., those corresponding to user steps). Filtering can remove certain frequencies (certain parts of the spectra we created in the previous exercise) and is often a necessity. To those unfamiliar with filters, the resource [8] is useful.

☞ **4.7**

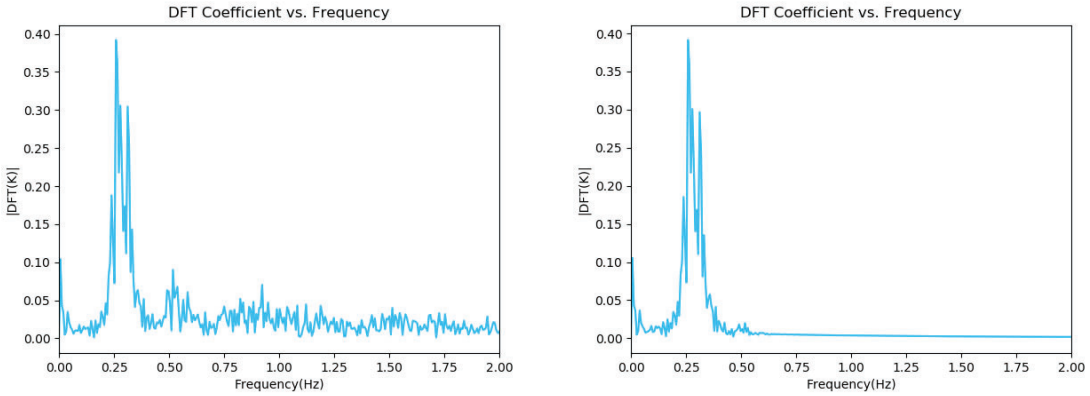
(a) Which frequencies (high or low) do you expect to be present in the 6 activities of this dataset? Use the answer on this question to determine sensible cutoff frequencies for your filters.



(a) Plot of test signal

(b) DFT of test signal

Figure 4.1: Example of a test signal and its DFT.



(a) No filtering on *Walking* data

(b) Low-pass filtering on *Walking* data

Figure 4.2: Example of low-pass filtering

(b) Apply a low-pass, a high-pass and a band-pass filter to the raw data. Report which of the filters is better for our data and what settings you have used. The Python `scipy` package already supports (Butterworth) filters, for these it is important that you set the right sampling frequency and keep in mind the Nyquist frequency. The order of the filter determines the slope of the filters envelope. Hint: you can use spectral plots created by the Fourier transform from the previous exercise to visualise how your filter functions and how steep the filter's slope is. See Figure 4.2 for an example of the spectra of a signal before and after low pass filtering (order 3 and cutoff frequency of 0.50 Hz).

□

4.2.6 Dynamic Time Warping and Classification with K-nearest Neighbours

Now that the dataset has been explored and the pre-processing of data has been handled, it is time to perform actual classification of the 6 activities. We cannot however just classify the raw signals or feature signals as is, since the signals differ in length and general structure. Even though the data from a walking activity is produced by walking, not all walking signals will be the same since everyone walks in a different manner and the signals may thus vary in time and speed. One way to measure the similarity between two time series is Dynamic Time Warping (DTW). For a very intuitive explanation you can turn to reference [9]. A Python Notebook by Punbatra [10] explains the algorithm with a hands on approach. The more theoretical background can be found in Chapter 4 of the book by Müller [11]. Once we have computed the similarity between signals, we can group or cluster the signals that are similar. These clustered signals likely belong to the same class (activity). A classifier can be trained on examples of signals from the train set and use

this information to determine what activity the signals in the test set belongs to. One classifier that can be used is the K-nearest Neighbour classifier. An intuitive explanation of the classifier is given in reference [12]. Some theoretical background can be found in Chapter 3.1 and 3.2 in the book by Murty [13]. Simple classifiers such as K-Nearest Neighbors can achieve high accuracy while using DTW as a distance metric for classifying/clustering time series data.

☞ **4.8** The tutorial in reference [14] applies DTW on derived features from the activity recognition data set. Apply the same technique but now on the raw data (all the raw data present in this dataset!). Include the classification results (the confusion matrices) for both the feature data and the raw data and reflect on the results and the differences. Also reflect on the ability to discriminate between the 6 activities. Note that the DTW algorithm uses brute force to compare all aspects of a signal. The complete classification of the tutorial can take very long (hours!) depending on your computer. It is okay if you leave on your laptop overnight, but if things take too long it is also fine to apply the classifier on a subset of the data. However, then you can expect that the performance in classification will be less (generally the more data the better) and you should mention this. The code allows you to play with the number of neighbours (parameter k) if you would like to see how this changes classification performance. □

4.2.7 Time Series Comparison and Prediction

For this assignment, we will switch to another dataset, namely the Kaggle Berkeley Earth Surface Temperature Study dataset. This dataset can be downloaded from reference [15]. This extensive data set is created by combining multiple existing archives and it allows groupings by country or city of interest. The dataset does not need an explanation as with the UCI HAR dataset and its usage is straightforward.

☞ **4.9**

(a) Plot the yearly temperatures for Norway, Finland, Singapore and Cambodia. Use DTW to measure the similarities between the temperature data of these countries and reflect on the results.
(b) Look at the tutorial on time series data by Aarshay Jain [16]. Use the Dickey-Fuller test to evaluate the stationarity of the temperature data for the four countries.
(c) Temperature and weather data includes seasons, day and night temperature changes as well as global warming. This seasonality and the slow trends (such as global warming) can be removed by differencing and decomposition techniques. Apply these techniques from the tutorial to de-trend the data and remove seasonality. Again apply DTW on your newly obtained processed data and reflect on the results.
(d) Read the section on *Forecasting models* in the tutorial and apply the AR, AM and ARIMA model on the temperature data of one of the four countries. Reflect on the results of the models on the data. Be clear in your methodology and explain which values for p, q and d you use based on the ACF and PACF plots.
(e) Select one of the models from the question above to make a temperature forecast (using for example the `forecast()` and `predict()` methods from the models) for the next seven days for your country of choice. You can take the next seven days from the last date present in the dataset of that country. Reflect on the quality of the prediction.

□

4.2.8 Basic tutorial on Python and common modules

We will cover basic concepts such as data types, output to the console, conditions, loops and simple examples from NumPy and Pandas packages. This tutorial is inspired by some of the tutorials already available [1,2,12] which can be referred for advanced tutorials on Python and relevant modules.

1. Let's get started by printing the following statement:
   The countdown begins , ... , 3 , 2 , 1 , 0.5 , Go!
   Assign the comma separated values to individual variables.

2. Arrays or (are) Lists:
   There is no native array data structure in Python but Lists which can support mixed data types [2].

Create an empty list and append the values from the individual variables created for the above question to the list.

3. Using operators on Strings and Lists:
   (a) How can we efficiently print repeating patterns such as, *blah blah blah blah blah blah blah blah blah blah*
   (b) Define 2 lists and join them by using '+' operator.
   (c) Repeat a list n times by using '*' operator.
4. IF statements, Boolean, in and is operators, and FOR loops:
   (a) Check if the value of the variable half used in question 1 is 0.5. If yes, print 'the value is correct', else, print 'the value is wrong'.
   (b) What is the output of the following code?

```
Half = 0.5
half = 0.5
if Half is half:
    print("they are the same instances")
elif Half == half:
    print("they have the same values")
```

   How can you make the if statement true?
   (c) Check if "Go!" is in the list created for question 2 and print the result. Hint: Use in operator.
   (d) Iterate over the individual elements in the list created for question 2 and printe each element.

   Note: Python uses indentation to define the code blocks. The agreed standard is 4 spaces, although tab or other number of space would also work.
5. Functions and modules:
   (a) Write a recursive function to print the first 'n' numbers in the Fibonacci sequence.
   (b) Python modules are nothing but simple .py files that contain executable statements and functions.
   A library can be installed using pip as follows: *! pip install pandas*
   A library can be imported by: *import pandas as pd*
6. NumPy arrays:
   NumPy is an extension for Python with support for array data structures and high-level operations on arrays.

```
# Create 2 new lists height and weight
height = [1.87,  1.87, 1.82, 1.91, 1.90, 1.85]
weight = [81.65, 97.52, 95.25, 92.98, 86.18, 88.45]

# Import the numpy package as np
import numpy as np

# Create 2 numpy arrays from height and weight
np_height = np.array(height)
np_weight = np.array(weight)
```

   Now we can perform element-wise calculations on height and weight. For example, you could take all 6 of the height and weight observations above, and calculate the BMI for each observation with a single equation. These operations are very fast and computationally efficient. They are particularly helpful when you have 1000s of observations in your data.

```
# Calculate bmi
bmi = np_weight / np_height ** 2

# Print the result
print(bmi)
```

Another great feature of Numpy arrays is the ability to subset. For instance, if you wanted to know which observations in our BMI array are above 23, we could quickly subset it to find out.

```
# For a boolean response
bmi > 23

# Print only those observations above 23
bmi[bmi > 23]
```

7. Pandas module:

The most popular data structure (atleast for data science) in Python is from Pandas called DataFrame which allows to represent and manipulate data in tabular form. For example, here is a dictionary containing details about BRICS countries.

```
dictionary = {"country": ["Brazil", "Russia", "India", "China", "South Africa"],
       "capital": ["Brasilia", "Moscow", "New Dehli", "Beijing", "Pretoria"],
       "area": [8.516, 17.10, 3.286, 9.597, 1.221],
       "population": [200.4, 143.5, 1252, 1357, 52.98] }

print(dictionary)
```

This dictionary can be simply converted into a dataframe as follows:

```
import pandas as pd
brics = pd.DataFrame(dictionary)
print(brics)

Now, various built-in functions can be used to
manipulate either the rows or columns.

print(brics.info())
print(brics.head())
print(brics.columns())
print(brics.values())
print(brics['population'])
print(brics.loc[:,['population','area']])
print(brics.sort_values(by='population'))
```

8. matplotlib:

The pyplot module from matplotlib provides MATLAB like interface to produce high-quality figures. A typical example is given below,

```
import numpy as np
import matplotlib.pyplot as plt

def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure(1)
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.savefig("path_to_file") # save figure
plt.show()
```

9. Reading and writing files:

Here are some examples for reading files,

```
import pandas as pd

df = pd.read_csv('path_to_file',header=None, sep='\s+')

df.columns = ['CRIM', 'ZN', 'INDUS', 'CHAS',
              'NOX', 'RM', 'AGE', 'DIS', 'RAD',
              'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
df.head()
```

The file path can be a url as well,

`https://raw.githubusercontent.com/rasbt/python-machine-learning-book/master/code/datasets/housing/housing.data`

Part II

Projects

Bibliography

[1] A beginners introduction to the Python Pandas package. http://pandas.pydata.org/pandas-docs/stable/10min.html Please copy the link, clicking may not work.

[2] C. Altın and O. Er, "Comparison of different time and frequency domain feature extraction methods on elbow gesture's emg," *European Journal of Interdisciplinary Studies*, vol. 5, no. 1, pp. 35–44, 2016. You can acces this article via this link: http://journals.euser.org/files/articles/ejis_may_aug_16/Cemil.pdf Please copy the link, clicking may not work.

[3] D. Jurafsky and J. H. Martin, *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition, 1st Edition*. Prentice Hall, Pearson Education International, 2009.

[4] William Cox provides an introduction to the Fourier transform and its implementation, the Fast Fourier Transform in his talk at the PyData Seattle conference in 2015. https://www.youtube.com/watch?v=YEwIjyOKFQ4 Please copy the link, clicking may not work.

[5] Chapter on Fourier Transforms by Sanjeev R. Kulkarni. The chapter can be viewed by the following link and then clicking on the document named "frequency.pdf": https://www.princeton.edu/~cuff/ele201/kulkarni_text/ Please copy the link, clicking may not work.

[6] The first section, titled "One dimensional discrete Fourier transforms" covers the implementation of Python's fft method and how to correctly plot the resulting spectrogram https://docs.scipy.org/doc/scipy/reference/tutorial/fftpack.html Please copy the link, clicking may not work.

[7] Nipun Batra gives a more practical approach to the FFT in this Python example. https://github.com/barleyj/ProgramaticallyUnderstandingSeries Please copy the link, clicking may not work.

[8] Explanation on audio filtering presented by Wick van den Belt for WickieMedia audio tutorials https://www.youtube.com/watch?v=rkwS6vigSyE Please copy the link, clicking may not work.

[9] Explanation of DTW presented by Thales Sehn Körting https://www.youtube.com/watch?v=_K1OsqCicBY Please copy the link, clicking may not work.

[10] Nipun Batra gives a more practical approach to the DTW algorithm in this Python example https://github.com/nipunbatra/ProgramaticallyUnderstandingSeries/blob/master/dtw.ipynb Please copy the link, clicking may not work.

[11] The book Information Retrieval for Music and Motion by Meinard Müller. You can read Chapter 4 for the information about Dynamic Time warping. The book is accessible with your student account via the following link. https://link-springer-com.ezproxy2.utwente.nl/content/pdf/10.1007%2F978-3-540-74048-3.pdf Please copy the link, clicking may not work.

[12] Video on KNN classification by Augmented Startups https://www.youtube.com/watch?v=MDniRwXizWo Please copy the link, clicking may not work.

[13] The book Pattern Recognition An Algorithmic Approach by Narasimha Murty. You can read section 3.1 and 3.2 for the information about the K-nearest Neighbour classifier. The book is accessible with your student account via the following link. https://link-springer-com.ezproxy2.

utwente.nl/content/pdf/10.1007%2F978-0-85729-495-1.pdfPlease copy the link, clicking may not work.

[14] Python tutorial on DTW and KNN classification of the UCI Har dataset by Mark Regan. https://github.com/markdregan/K-Nearest-Neighbors-with-Dynamic-Time-Warping/blob/master/K_Nearest_Neighbor_Dynamic_Time_Warping.ipynb Please copy the link, clicking may not work.

[15] Berkeley EarthSurface Temperature Study dataset. Accessible via the link below. https://www.kaggle.com/berkeleyearth/climate-change-earth-surface-temperature-data Please copy the link, clicking may not work.

[16] Time Series Modelling and Forecastig by Aarshay Jain. Tutorial and Python code can be accessed through the link below. https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/ Please copy the link, clicking may not work.

[17] B. Boashash, *Time-frequency signal analysis and processing: a comprehensive reference*. Academic Press, 2015.

[18] B. Greene, S. Faul, W. Marnane, G. Lightbody, I. Korotchikova, and G. Boylan, "A comparison of quantitative eeg features for neonatal seizure detection," *Clinical Neurophysiology*, vol. 119, no. 6, pp. 1248–1261, 2008.

[19] T. E. Inder, L. Buckland, C. E. Williams, C. Spencer, M. I. Gunning, B. A. Darlow, J. J. Volpe, and P. D. Gluckman, "Lowered electroencephalographic spectral edge frequency predicts the presence of cerebral white matter injury in premature infants," *Pediatrics*, vol. 111, no. 1, pp. 27–33, 2003.

[20] J. W. Sleigh, E. Olofsen, A. Dahan, J. De Goede, and D. A. Steyn-Ross, "Entropies of the eeg: the effects of general anaesthesia," 2001.

[21] M. Haenggi, H. Ypparila-Wolters, C. Bieri, C. Steiner, J. Takala, I. Korhonen, and S. M. Jakob, "Entropy and bispectral index for assessment of sedation, analgesia and the effects of unpleasant stimuli in critically ill patients: an observational study," *Critical Care*, vol. 12, no. 5, p. 1, 2008.

[22] R. Shalbaf, H. Behnam, J. Sleigh, and L. Voss, "Measuring the effects of sevoflurane on electroencephalogram using sample entropy," *Acta Anaesthesiologica Scandinavica*, vol. 56, no. 7, pp. 880–889, 2012.

[23] Z. Liang, Y. Wang, X. Sun, D. Li, L. J. Voss, J. W. Sleigh, S. Hagihira, and X. Li, "EEG entropy measures in anesthesia," *Frontiers in computational neuroscience*, vol. 9, p. 16, 2015.

[24] J. Bruhn, L. E. Lehmann, H. Röpcke, T. W. Bouillon, and A. Hoeft, "Shannon entropy applied to the measurement of the electroencephalographic effects of desflurane," *The Journal of the American Society of Anesthesiologists*, vol. 95, no. 1, pp. 30–35, 2001.

[25] E. Olofsen, J. Sleigh, and A. Dahan, "Permutation entropy of the electroencephalogram: a measure of anaesthetic drug effect," *British journal of anaesthesia*, vol. 101, no. 6, pp. 810–821, 2008.

[26] X. Nan and X. Jinghua, "The fractal dimension of eeg as a physical measure of conscious human brain activities," *Bulletin of Mathematical Biology*, vol. 50, no. 5, pp. 559–565, 1988.